How Can We Conduct "Fair and Consistent" Hardware Evaluation for SHA-3 Candidate?

Shin'ichiro Matsuo[1], Miroslav Knežević[2], Patrick Schaumont[3], Ingrid Verbauwhede[2], Akashi Satoh[4], Kazuo Sakiyama[5] and Kazuo Ota[5]

[1] National Institute of Information and Communications Technology, Japan
[2] Katholieke Universiteit Leuven, Belgium
[3] Virginia Tech, USA
[4] National Institute of Advanced Industrial Science and Technology, Japan
[5] The University of Electro-Communications, Japan

Abstract. The objective of the NIST SHA-3 competition is to select, among multiple competing candidates, a standard algorithm for cryptographic hashing. The selected winner will need to have adequate cryptographic properties and good implementation characteristics over a wide range of target platforms including both software and hardware. The performance evaluation in hardware is particularly challenging. In technical sense, the reasons are the large design space, the wide range of target technologies and the multitude of optimization criteria. The effort for completing the evaluation for all candidates is heavy. Moreover the evaluation criteria must be consistent and fair in the sense of management of open competition. In this contribution we describe the efforts of five research groups to evaluate SHA-3 candidates using a common prototyping platform. Using a SASEBO-GII FPGA board as a starting point, we evaluate the performance of the 14 remaining SHA-3 candidates with respect to area, throughput and power consumption. Our approach defines a standard testing harness for SHA-3 candidates, including the interface specification for the SHA-3 module on the SASEBO testing board.
Keywords: Hash Function, SHA-3 and Hardware Evaluation

1 Introduction

We briefly explain the origin of the NIST SHA-3 competition, and the need for a standard hardware evaluation mechanism.

1.1 Background

Since collisions on standard hash functions were reported in 2004, improvements to hash attack methods and improvements to hash algorithms have been at a similar, rapid pace [1]. For this reason, NIST decided to initiate development of a new hash standard. They use a competition model for this development [2], similar to the development of the present block cipher standard, AES.

The competition is organized in three phases, with the second phase scheduled to complete in the summer of 2010. Out of the original 64 submissions to the first phase, 14 candidates have been selected for detailed analysis in the second phase. NIST will then reduce this set to an even smaller number during the third, final phase.

The selection of winning candidates is driven by considering security as well as implementation efficiency of the proposed hash algorithms in hardware and software. However, systematic cryptanalysis of hash functions is not well established, and it is hard to measure the cryptographic strength of a hash function beyond obvious metrics such as digest length. For this reason the implementation efficiency of hardware and software plays a vital role in the selection of the finalist.

There are several ongoing projects that evaluate the hardware efficiency of the SHA-3 candidates [3–6], however, the validity and consistency of the evaluation criteria and methods of such research are not well discussed yet. In order to evaluate the hardware efficiency over a set of SHA-3 candidates, we need to fix an evaluation environment (*i.e.* platform), and implementation method (*i.e.* design strategy), and a performance comparison method (*i.e.* evaluation criteria). The consensus on such points is required for a fair comparison.

Performance evaluation of hardware, including the measurement of power consumption, execution time, and hardware resources, is a complicated problem. There are several reasons for this. Most importantly, the design space for hardware performance evaluation is larger than for software performance evaluation. Additional design constraints (such as low-area, max-throughput, and min-energy) are required to define an optimum implementation. Second, accurate and generic performance evaluation metrics are hard to obtain. Throughput can be characterized provided that the hardware design can be accurately timed. Area metrics depend strongly on the target technology (ASIC/FPGA). Power consumption is the most difficult, and it is almost never mentioned in publications.

Because there are many possible application scenarios for hash functions, a universal set of hardware evaluation criteria may be hard to define. However, in this paper, we therefore focus on criteria that are consistent and fair on a given platform.

1.2 Related Work

Recently, several research groups have proposed a comprehensive performance evaluation methods, which evaluate a set of hash algorithms on a common platform. These proposals use a synthesis target.

- Tillich *et al.* developed RTL Verilog code for all SHA-3 candidates. They present syntheses results on UMC 180nm technology [3].
- Gaj *et al.* developed a scripting system called ATHENA, targeted towards FPGA [7]. A fair comparison is achieved by defining a standard interface and by automatic design space exploration.
- Baldwin *et al.* [8] propose several hash candidates, as well as a standard interface to achieve a fair comparison.

1.3 Our Contribution

The main contributions of the paper can be summarized as follows. First, we reconsider requirements and concepts for hardware evaluation of the hash functions. To select the final SHA-3 algorithm from the second-round candidates, we need to have a selection criteria that covers possible hash applications, a technical accuracy and a fair comparison after all. We describe the details of such criteria in Section 2.

Second, we propose a consistent evaluation scheme for the hardware implementation. We evaluate all of the 14 second-round candidates with the evaluation criteria which fulfills the above requirements. We propose a platform, a design strategy, and an evaluation criteria (evaluation items and metrics) for a fair hardware evaluation of the SHA-3 candidates. In comparison with the efforts described above, we use a prototyping approach rather than a synthesis for ASIC. We map each of the hash candidates onto a SASEBO-GII FPGA board [9]. We then evaluate the hash candidates with respect to throughput, latency, hardware cost and power consumption.

2 Requirements for Hardware Evaluation

In this section, we reconsider the requirements of hardware evaluation for the international standard of hash function. There are three main requirements to be considered.

First, we review the ISO and IETF standards and outline the usages and platforms we need to take into account. Second, we consider the accuracy of the evaluation. There are many possible viewpoints on the hardware evaluation of cryptographic algorithms. We choose important viewpoints according to the specific hash applications, and then show the necessary technical aspects.

Last but not least, a very important requirement to be considered is a fair evaluation. We need to set the evaluation criteria such that it provides non-biased results and is fair for all the candidates. To do so, we need to define a unified evaluation criteria (set the platform, method, etc), and make the results publicly available for further verification. Next, we describe the above requirements in detail.

2.1 Coverage of Hash Usage

Hash functions are used in many cryptographic algorithms and protocols. In case of cryptographic algorithms, hash functions are used in digital signature scheme, message padding of public key encryption scheme and message authentication code. To illustrate, we mention some of the cryptographic algorithms that utilize a hash function in the ISO standard.

- Public key encryption scheme (ISO 18033).
- Message authentication code (ISO 9797).
- Signature scheme giving message recovery (ISO 9796).

– Signature scheme with appendix (ISO 14888).
– Authenticated encryption (ISO 19772).

In case of cryptographic protocols, hash functions are used in multiple scenarios, as follows.

– Establishing secure channels such as SSL (RFC 2246, 3546, etc), SSH (RFC 4523) and IPsec (RFC 4109 and 2406): Hash functions are used in authentication by public key certificate, key agreement, integrity check and private communication. In most cases, such functionalities are executed on servers and PCs. However, a use of a smart-card based authentication by a public key certificate is also possible.
– Entity authentication (ISO 9798), Kerberos (RFC 4120), EAP (IEEE 802.1X) and APOP (RFC 2195 and 2095): Hash functions are used in challenge-response protocols, as well as a part of digital signature and HMAC.
– Key exchange protocol (ISO 11770): Hash functions are used for randomizing secret information.
– Public key infrastructure (RFC 3280, 4325): Hash functions are used for issuing and verifying public key certificates.
– Digital time stamping (ISO 18014 and 3, RCC 3161): Hash functions are used for calculating time-stamp tokens.
– Secure e-mail (S/MIME RFC 2311, 2312, 3850 and 3851, PGP RFC 3156): Hash functions are used for issuing and verifying public key certificates.
– One-time password: Hash functions are used to calculate one-time password. Some of the one-time password products are implemented in hardware tokens as well.

The use of the hash functions outlined above is mainly considered as a software implementation on a PC (desktops, servers, etc). To have a complete overview of the hash usage, now mainly in the content of a hardware implementation, we wrap up the list with the following two items.

– Hardware Security Module (HSM): Tamper-resistant cryptographic module. HSM is used to issue public-key certificates with protecting private key of the certificate authority.
– Smart-card: Smart-cards are widely used for authentication and signing data.

Outlined above, are just some of the application specific scenarios of a hash usage. The list is not complete and serves to illustrate the importance of a hash algorithm in cryptographic algorithms and protocols.

2.2 Consistency and Accuracy in Hardware Evaluation

For a complete comparison of hardware characteristics, we need to consider following hardware figures.

– Speed performance (frequency, number of cycles, in short – processing speed).

– Size of a circuit (equivalent gate count, number of utilized slices, LUTs, etc).
– Power consumption (at certain fixed frequency).

A consistent evaluation environment for all SHA-3 candidates needs to be prepared in that case. To realize such an environment, we need to fix the architecture of hardware implementation, common platform (FPGA/ASIC implementation) and fix the interface specification for measurement. We also need to define evaluation items and metrics for the actual measurement. Naturally, these metrics and their accuracy need to meet a technical specification of the target hardware.

2.3 Fair Comparison

The important requirement for an open competition such as the SHA-3 competition is a fair comparison. To achieve the goal, we need to consider the following two aspects.

First, the evaluation environment needs to be open and available to all designers and evaluators. It also needs to be unified and common for all the candidates. Second, the claimed results need to be reproducible and open for a public verification. By using a common, fixed platform and making our code publicly available, we achieve a desired goal.

3 Overall Evaluation Project

By taking into account a large number of second-round candidates, a design space exploration and the hardware figures necessary for a complete comparison, it is obvious that a collaboration between several research groups provides a good environment and a good starting point for conducting a fair and a consistent hardware evaluation of the SHA-3 candidates. Therefore, we organize the overall evaluation project as a collaboration between five different research teams. Such an approach opens new issues in addressing the management of the whole project. A good coordination between the research teams is required in order to reduce duplicative efforts as well as for a consistent measurement results.

As mentioned in Section 2.2, there are three main hardware figures that need to be considered when comparing different designs. Therefore, a couple of major constraints need to be satisfied. First, as mentioned in Section 2.3, the evaluation environment needs to be open and available to all designers and evaluators. In addition, the RTL code of all SHA-3 candidates needs to be available for a public verification. Following that approach, the claimed results become reproducible and open to the whole community.

Next, we discuss our proposed evaluation scheme. We describe the evaluation environment, hardware/software interface, design strategy, evaluation metrics and finally, we provide the experimental results.

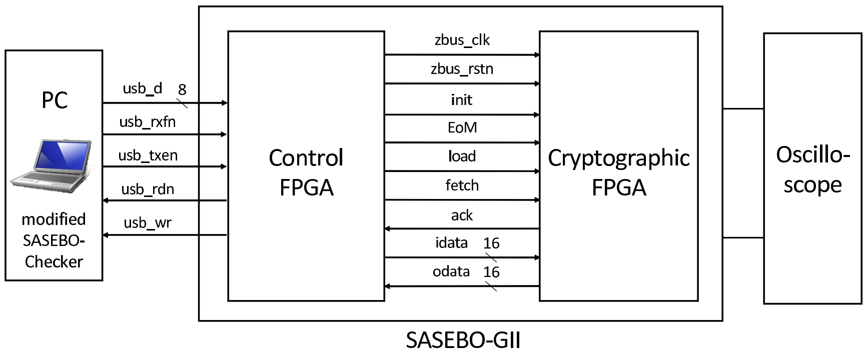

Fig. 1. Evaluation Environment Using SASEBO-GII.

3.1 Our proposed evaluation scheme

Fixing evaluation platform Figure 1 illustrates the target platform for our evaluation, which includes a SASEBO-GII board and a PC. The SASEBO board includes two FPGA: a control-FPGA and a cryptographic-FPGA. On the PC, a test program enables a user to enter a sample message, which is transmitted to the control FPGA through USB. The control FPGA controls the data flow to send this message to the cryptographic FPGA, where hash operations are performed. After the hash operations, the digest is returned to the PC through the control FPGA. As illustrated in Fig. 1, the interface between the control FPGA and the cryptographic FPGA is fixed and common among all SHA-3 candidates.

The control FPGA checks the latency of a single hash operation for an input data that is performed in the cryptographic FPGA and reports the number of clock cycles to the PC. The PC then reports two different performance metrics. One is the number of clock cycles including the cycles for receiving input data and the other is one excluding the cycles for the data input.

During hashing of a message, we also measure the power of the hashing operation. This trace, in combination with performance data, enables a precise characterization of the power dissipation and energy consumption of the SHA-3 candidate on the cryptographic FPGA.

Hardware and Software Interface A key concept in our approach is the use of a standard interface to integrate hash algorithms inside of the cryptographic FPGA. In this section, we describe the major principles of this interface. We also compare our ideas with those of several other proposals, including the interfaces defined by Chen *et al.* [10], by Gaj *et al.* [7], by Kobayashi *et al.* [11], and by Baldwin *et al.* [8].

In the following observations, it is useful to refer to the method used to interface SHA-3 candidates in *software*. The software uses an *API* or Application Program Interface. For hash algorithms, three function calls are used.

- `void init(hashstate *d)` initializes the algorithm state of the hash, which is typically stored in a separate structure in order to make the hash implementation re-entrant.
- `void update(hashstate *d, message *s)` hashes a message of a given length and updates the hash state. The message is chopped into pieces of a standard length called a *block*. In case the message length is not an integral number of blocks, the API will use a *padding* procedure which extends the message until it reaches an integral number of blocks in length.
- `void finalize(hashstate *d, digest *t)` extracts the actual digest from the hash state.

A hardware interface for a SHA-3 module will emulate similar functionality as the software API interface. The hardware interface will therefore need to address the following issues.

Handshake protocol: The hash interface needs to synchronize data transfer between the SHA-3 module and the environment. This is done using a handshake protocol and one can distinguish a *master* protocol from a *slave* protocol, depending on which party takes the initiative to establish synchronization. The interfaces by Chen and Kobayashi use a slave protocol for the input and the output of the algorithm. The interfaces by Baldwin and Gaj define a slave protocol for the input and a master protocol for the output. The former type of interface is suited for a co-processor in an embedded platform, while the latter type of interface is suited for high-throughput applications that integrate the SHA module to FIFO's. The interface in our proposal uses a slave protocol.

Wordlength: Typical block and digest lengths are wider (*e.g.* 512 bits) than the word length that can be provided by standard platforms (*e.g.* 32 bits), so that each hash operation will result in several data transfers. While this overhead is typically ignored by hardware designers, it is inherently part of the integration effort of the SHA-3 module. All of the interface proposals leave the standard interface word length undefined, although they implicitly assume 32 bits. In our proposal, we use a 16-bit interface, which is defined by the data-bus between the control FPGA and the cryptographic FPGA.

Control: The functions of the software API need to be translated to equivalent hardware control signals. One approach, followed by Gaj, is to integrate this control as in-band data in the data stream. A second approach is to define additional control signals on the interface, for example to indicate message start and end. This is the approach taken by Chen, Kobayashi, and Baldwin. We follow the same approach in our proposal as well.

Padding: Finally, *padding* may or may not be included in the SHA-3 hardware module. In the latter case, the hardware module implicitly assumes that an integral number of blocks will be provided for each digest. Common padding schemes are defined by in-band data formatting, and this makes it possible to implement padding outside of the hardware module. The interface proposal by Baldwin explicitly places padding hardware into the interface. The other interface proposals leave padding to the SHA-3 designer. However, Chen and Kobayashi assume hardware padding will only be implemented at word-level,

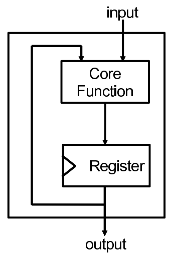

Fig. 2. Our design strategy (Fully Autonomous)

while Gaj supports bit-level padding as well. We follow the approach of Chen and Kobayashi.

Note that there are many possible solutions to the interface problem, and that we present one possible approach. We observe that the key issue for a fair comparison is to use a *common* interface for all candidates. In addition, we will show that our performance evaluation mechanism allows to factor out the overhead of the interface communication.

Design Strategy Besides a standard platform, our approach also defines a design strategy.

There are mainly three types of architectures, fully autonomous, external memory and core functionality, which are implemented on the cryptographic FPGA [12]. And we selected fully autonomous for fair evaluation. In this architecture(Fig. 2), one transfers message data to a hash function over multiple clock cycles, until a complete message block is provided. The hash module buffers a complete message block locally, before initiation the hash operation. Therefore, this architecture can work autonomously, and the resulting hash module is well suited for integration into other architectures, such as system-on-chip.

The previous work for evaluating hardware performance has been executed without using a standardized architecture, *i.e.* different architectures are used. For example, the design method by Namin *et al.* [4] and Baldwin *et al.* [5] is based on the core functionality type and they evaluate a rough estimate of the performance of hash function hardware. On the other hand, the design method by Tillich *et al.* [3] and Jungk *et al.* [6] is based on the fully-autonomous type. They assumed that the input data for the hash function hardware is sent in one cycle, so that the length of the input data is assumed long (*e.g.* 256 or 512 bits). Consequently, their evaluation results cannot be used straightforwardly for a performance evaluation of an accelerator of CPU where only a limited size of data access is available in one.

In this paper, we evaluate the performance of SHA-3 candidates when they are used in a real system. In addition, we use performance metrics that enable us to separate the interface operation from the hash operation. On the cryptographic FPGA of SASEBO-GII, we use the approach shown in Fig. 2. Compared to external memory, this approach avoids off-chip memory access, and it also avoids the idealized (unrealistic) interface utilized by core functionality.

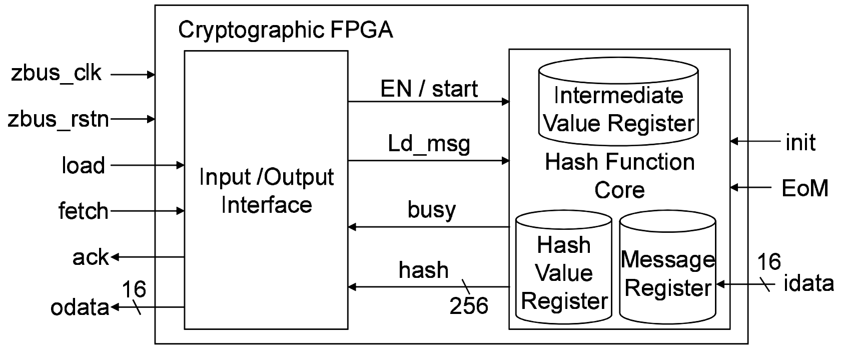

Fig. 3. Architecture of Cryptographic FPGA.

Figure 3 shows the detailed architecture of the cryptographic FPGA which we use for evaluating hardware performance. The cryptographic FPGA consists of an interface block which controls input and output, and a core function block which executes a hashing process. There are several SHA-3 candidates which need to keep an input message during the hashing process. In our environment, it is able to prepare a message register in the core function block. This architecture can reuse the message register for next message block. Also, we may be able to prepare an output register in the core function block which keeps the result (*e.g.* 256-bit hash value). We used output register for all second-round candidates in this paper.

How we measure the hardware evaluation The third component of our evaluation strategy is the performance evaluation criteria. These are explained in this section.

Evaluation Items We implement fourteen SHA-3 hash candidates on the cryptographic FPGA, Xilinx Virtex-5 (xc5vlx30-3ff324) on SASEBO-GII. We check the hardware performance in terms of speed and hardware cost. The speed performance is evaluated with its latency or throughput that is calculated with the input block size, the maximum clock frequency, and the total number of clock cycles with or without the communication overheads. The cost performance is evaluated with the number of slices, registers and LUTs. A hash function which has a high throughput with a low hardware cost is regarded as efficient. The power consumption of the hash design is measured during a complete hash operation. The energy cost is the integral of the power consumption over the period of hash operation. The energy cost is normalized to the message block length, in order to obtain a standardized nJ/bit metric.

Speed Performance Metrics We use the following notations to explain the criteria.

$$\begin{aligned}
B &: \text{Input block size,}\\
I &: \text{Total number of clock cycles,}\\
D &: \text{Critical path delay,}\\
Th &: \text{Throughput,}\\
f_{max} &: \text{Maximum clock frequency,}\\
M &: \text{The size of the message without padding,}\\
M_p &: \text{The size of the message with padding.}
\end{aligned}$$

Table 1. Evaluation Metrics.

	Long Message (Throughput)	Short Message (Latency)
Interface + Core	$\frac{B \cdot f_{max}}{I_{in}+I_{core}}$	$\frac{M_p}{Th}$
Core Function Block	$\frac{B \cdot f_{max}}{I_{core}}$	$\frac{M_p}{Th_{core}}$

A hash function executes a hashing process for each data with an input block size, and uses the result as the next input data to proceed the whole hashing process. The clock cycles necessary for hashing M-bit data can be expressed as

$$I = \frac{M_p}{B}(I_{in} + I_{core}) + I_{final} + I_{out} \ . \tag{1}$$

Here, $\frac{M_p}{B}$ is the number of hash core operations when the hash core can perform B-bit data in one operation. I_{in}, I_{core}, I_{final} and I_{out} denote the number of clock cycles used to input data, to execute hashing process in the core function block, to perform the final calculation process and to output the hash results, respectively. Note that the coefficients of I_{final} and I_{out} are both ones, because these processes are only executed when outputting the resultant data.

As a result, the throughput and the latency can be expressed as

$$Th = M_p \times \frac{f_{max}}{\frac{M_p}{B}(I_{in} + I_{core}) + I_{final} + I_{out}}, \tag{2}$$

$$L = \frac{M_p}{Th} \ . \tag{3}$$

When M_p is sufficiently large, for example in the case of hashing a long message, I_{final} and I_{out} can be negligible from Eq. (2). In this case, the throughput $Th_{LongMessage}$ is approximated as

$$Th_{LongMessage} = \frac{Bf_{max}}{I_{in} + I_{core}}. \tag{4}$$

On the other hand, when M_p is small, for example in the case of hashing a short message for authentication, we cannot ignore I_{final} and I_{out}. Moreover, as the latency is an important metrics for a short message rather than the throughput, we use Eq. (3) to compare the speed performance of the SHA-3 candidates.

Table 1 shows the evaluation metrics. Here, the throughput of the core function block Th_{core} is

$$Th_{core} = M_p \times \frac{f_{max}}{\frac{M_p}{B} I_{core} + I_{final}}. \tag{5}$$

3.2 Evaluation results

In this work, we implement SHA-256 and all second-round SHA-3 candidates aiming at a high-speed hardware implementation [6]. Although it is not possible to completely factor out the designer in our comparison, the 15 algorithms were all prototyped and tested using the same evaluation platform. Each of them was evaluated according to the metrics indicated above, covering performance, area, power consumption and energy consumption.

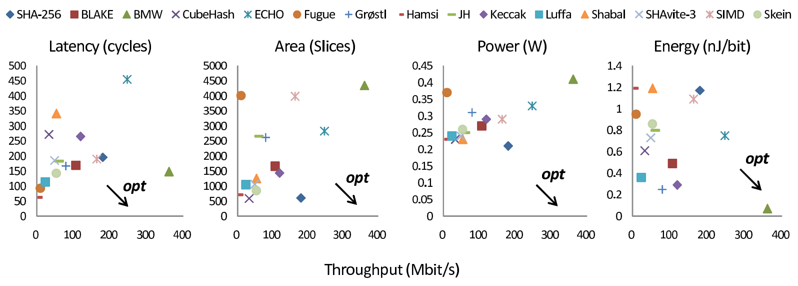

Fig. 4. Throughput vs Latency, Area, Power, and Energy. Pareto points (excluding SHA-256) are as follows. Throughput/Latency - BMW, Luffa, Fugue, Hamsi. Throughput/Area - ECHO, Keccak, Shabal, Skein, CubeHash. Throughput/Power - BMW, ECHO, SIMD, BLAKE, Shabal. Throughput/Energy - ECHO, Keccak, Grøstl.

[6] We plan to release the Verilog/VHDL source code for these 15 algorithms.

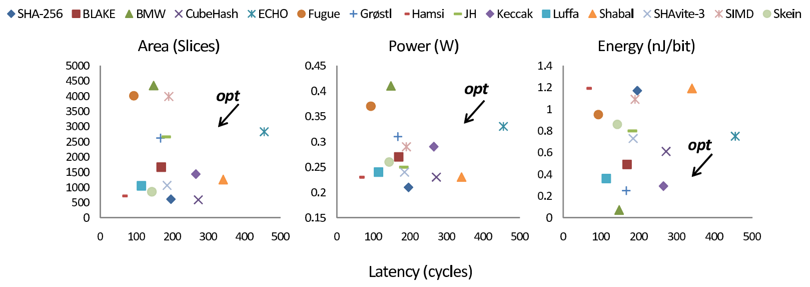

Fig. 5. Latency vs Area, Power, and Energy. Pareto Points (excluding SHA-256) are as follows. Latency/Area - Hamsi, CubeHash. Latency/Power - Hamsi, CubeHash, Shabal. Latency/Energy - Fugue, Luffa, BMW.

Table 2 shows a comprehensive summary of the measurement results. As with all measurement data, it is important to understand the assumptions used when collecting these numbers. The table includes the following quantities for each candidate.

- The message block size;
- The highest clock frequency achievable with the design on a Virtex-5 FPGA.
- The latency in terms of clock cycles. Several cases are shown: the cycle count of the core function and the input interface overhead; the cycle count of the complete design; the cycle count of the core function; the cycle count of the core function including final processing.
- The throughput of the design in clock megabit per second. This number assumes that the FPGA is operating at the maximum achievable clock frequency for the given design. Both the throughput with and without interface-overhead is shown.
- The latency of the design for short messages, in microseconds. This number assumes that the FPGA is operating at the maximum achievable clock frequency for the given design. Both the latency with and without interface-overhead is shown.
- The area cost of the design, in terms of occupied Virtex-5 slices, number of flip-flops, and number of LUTs.
- The power consumption of the design for long and short messages. For long messages, the average power consumption includes only the core functionality. For short messages, the average power consumption includes the core functionality and the finalization. The power consumption is measured directly on the core power supple (V_{vccint}) of the FPGA. The power consumption is measured with the FPGA operating at 24 MHz.
- The energy consumption of the design for long and short messages. The energy consumption is normalized against the block size, and expressed in nJ/bit. Also in this case, the difference between long-message energy and

short-message energy relates to inclusion of the finalization processing in the measurement.

We measured the static power dissipation of the V5 FPGA on Sasebo-GII to be around 200mW. Hence, the power numbers listed in Table 2 are dominated by static power dissipation rather than dynamic power dissipation. One should keep in mind that static power dissipation depends on the FPGA, while dynamic power dissipation depends on the SHA3 candidate. Thus, even though we report 'total power', there is a significant impact from the FPGA component.

3.3 Open issues

The analysis of this data must be done carefully, as there are many different dimensions to consider. In Figs. 4 and 5, we investigate the connection between Throughput and Latency/Area/Power/Energy, and between Latency and Area/Power/Energy. We recommend to make either a color-print of these graphs, or else study the figures on a color screen. The plots were made using the data from Table 2; but the throughput-numbers for each design were normalized to 24 MHz to make them comparable with the power- and energy numbers. The optimum for the plots in Fig. 4 lies in the lower-right corner. The optimum for the plots in Fig. 5 lies in the lower-left corner.

Each plot can now be evaluated using Pareto-analysis. The *Pareto* points of each plot are those points which simultaneously outperform other points in two cost factors. For example, in the Latency-vs-Throughput plot (left plot of Fig. 4), the BMW design has a higher throughput and a lower latency than most other designs, except for Hamsi, Fugue and Luffa, which have a lower latency (but a lower throughput). The plots demonstrate that the Pareto points can change for each pair of criteria. This confirms the richness of the design space of SHA-3 candidates.

While we cannot provide a conclusive analysis of this data at this moment, we can make some general observations. There is a group of designs that have related characteristics, and appear as a cluster in the graphs. These include Hamsi, Luffa, CubeHash, Skein, Shabal, SHAvite-3, Keccak, Grøstl and BLAKE. The other designs have outlier-characteristics in one or more aspects. These designs include ECHO, BMW, SIMD, JH, and Fugue. We plan further analysis of these algorithms with respect to the proposed evaluation criteria.

4 Conclusion

For a complete hardware evaluation, there are plenty of evaluation platforms to be considered. Therefore, fixing an evaluation platform is crucial for conducting a fair and a consistent comparison. In this paper, we propose an evaluation platform and a consistent evaluation method to conduct a fair hardware evaluation of the remaining SHA-3 candidates. This proposal meets the requirements analyzed from actual hash applications and conditions of standard selection. The

Table 2. Results of the SHA-3 Candidates on Virtex-5 (xc5vlx30-3ff324).

Implementation	Input Block Size [bits]	Max. Clock Freq [MHz]	Total Number of Clock Cycles [cycles]		Long Message Throughput [Mbps]	Short Message M=1,024 Latency [μs]	Number of Occupied Slices	Number of Slice Registers	Number of Slice LUTs	Power [W]		Energy [nJ/bit]	
			I/F + Core	Core						Long Msg	Short Msg	Long Msg	Short Msg
SHA-256	512	260	148 (196)	68 (68)	899 (1,958)	1.89 (0.785)	609	1,224	2,045	0.21	0.21	1.17	1.17
BLAKE-32	512	115	121 (169)	22 (22)	487 (2,676)	3.57 (0.574)	1,660	1,393	5,154	0.27	0.27	0.49	0.49
BMW-256	512	34	98 (148)	2 (4)	178 (8,704)	10.12 (0.235)	4,350	1,317	15,012	0.41	0.41	0.07	0.09
CubeHash16/32-256	256	185	64 (272)	16 (176)	740 (2,960)	2.85 (1.297)	590	1,316	2,182	0.23	0.23	0.61	1.82
ECHO-256	1,536	149	407 (455)	99 (99)	562 (2,312)	3.05 (0.664)	2,827	4,198	9,885	0.33	0.33	0.89	0.89
Fugue-256	32	78	8 (93)	2 (39)	312 (1,248)	4.47 (1.321)	4,013	1,043	13,255	0.36	0.37	0.95	1.47
Grøstl-256	512	154	106 (164)	10 (20)	744 (7,885)	2.44 (0.260)	2,616	1,570	10,088	0.31	0.31	0.25	0.25
Hamsi-256	32	210	10 (63)	4 (9)	672 (1,680)	1.92 (0.690)	718	841	2,499	0.23	0.23	1.19	1.27
JH-256	512	201	135 (183)	39 (39)	762 (2,639)	2.25 (0.582)	2,661	1,612	8,392	0.25	0.25	0.80	0.80
Keccak(-256)	1,024	205	217 (265)	25 (25)	967 (8,397)	2.35 (0.244)	1,433	2,666	4,806	0.29	0.29	0.29	0.29
Luffa-256	256	261	57 (114)	9 (18)	1,172 (7,424)	1.31 (0.207)	1,048	1,446	3,754	0.24	0.24	0.36	0.43
Shabal-256	512	228	143 (341)	50 (200)	816 (2,335)	2.75 (1.316)	1,251	2,061	4,219	0.23	0.23	1.19	2.15
SHAvite-3_{256}	512	251	134 (185)	38 (38)	959 (3,382)	1.79 (0.454)	1,063	1,363	3,564	0.24	0.24	0.73	0.73
SIMD-256	512	75	142 (190)	46 (46)	270 (835)	6.32 (1.840)	3,987	6,693	13,908	0.29	0.29	1.09	1.09
Skein-256-256	256	115	75 (143)	21 (41)	393 (1,402)	3.20 (0.904)	854	929	2,864	0.26	0.26	0.86	1.08

platform includes a SASEBO-GII evaluation board, evaluation software, and appropriate interface definition. Using this method, we implement all second-round SHA-3 candidates and obtain the resulting cost and performance factors. This technical study provides a fair and a consistent evaluation scheme. We hope that our experience helps future SHA-3 evaluation and a similar selection of standard cryptographic algorithms.

References

1. W. E. Burr, "Cryptographic Hash Standards: Where Do We Go from Here?," IEEE Security & Privacy, Vol. 4, no 2, p. 88-91, 2006.
2. National Institute of Standards and Technology (NIST), "Cryptographic Hash Algorithm Competition," http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.
3. S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J. Schmidt, and A. Szekely, "High-Speed Hardware Implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein," IACR ePrint archive, 2009/510, 2009.
4. A. H. Namin and M. A. Hasan, "Hardware Implementation of the Compression Function for Selected SHA-3 Candidates," CACR 2009-28, 2009.
5. B. Baldwin, A. Byrne, M. Hamilton, N. Hanley, R. P. McEvoy, W. Pan, and W. P. Marnane, "FPGA Implementations of SHA-3 Candidates:CubeHash, Grøstl, Lane, Shabal and Spectral Hash," IACR ePrint Archive, Report 2009/342, 2009.
6. B. Jungk, S. Reith, and J. Apfelbeck, "On Optimized FPGA Implementations of the SHA-3 Candidate Grøstl," IACR ePrint Archive, Report 2009/206, 2009.
7. CERG at George Mason University. Hardware Interface of a Secure Hash Algorithm (SHA). Functional Specification, October 2009.
8. B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. Marnane, "A Hardware Wrapper for the SHA-3 Hash Algorithms," IACR ePrint archive, 2010/124, 2010.
9. "National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Information Security (RCIS): "Side-channel Attack Standard Evaluation Board (SASEBO)"," http://www.rcis.aist.go.jp/special/SASEBO/SASEBO-GII-en.html.
10. Z. Chen, S. Morozov, and P. Schaumont, "A Hardware Interface for Hashing Algorithms," IACR ePrint archive, 2008/529, 2008.
11. K. Kobayashi, J. Ikegami, S. Matsuo, K. Sakiyama, and K. Ohta, "Evaluation of Hardware Performance for the SHA-3 Candidates Using SASEBO-GII," IACR ePrint archive, 2010/10, 2010.
12. ECRYPT II, "SHA-3 Hardware Implementations," http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations.